Wisconsin Standards for Computer Science

What is Computer Science Education?

Wisconsin defines Computer Science (CS) as an academic discipline that encompasses the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, networks, and their impact on society. The standards outlined in this document provide an important foundation to prepare students for post-secondary education and careers.

Computer Science Education in Wisconsin

Computer Science drives job growth and innovation throughout the economy and society. In 2017, demand for computing jobs in Wisconsin was higher than any other occupation category[1], and this growth is projected to continue for much of the next decade[2]. The need for CS education is increasing because all students will need some foundational knowledge in CS, regardless of their occupational path. To offer formal coursework and integrate CS into K-12 learning opportunities, developing CS academic standards across grades K-12 is an essential first step. In the 21st century, career and college readiness will increasingly require a CS component.

At the elementary level, CS content and concepts can be integrated throughout the curriculum. Teachers can effectively use CS concepts in instruction to develop foundational skills and also can create a connection to secondary CS options. At the middle and high school levels, all students should have access to CS, including those who wish to pursue advanced courses.

Wisconsin's Vision for Computer Science

The Wisconsin vision for CS is shaped by Wisconsin practitioners, experts, and the business community, and is informed by work at the national level and in other states. The overarching goal of Wisconsin's vision for CS is to introduce the principles and methodologies of

---

[1] Department of Workforce Development, State of Wisconsin, *Hot Jobs*, 2017. http://wisconsinjobcenter.org/labormarketinfo/

[2] Bureau of Labor Statistics, U.S. Department of Labor, Occupational Outlook Handbook, 2016-17 Edition. https://www.bls.gov/ooh/computer-and-information-technology/

CS to all students, whether they are college or career bound after high school. Wisconsin's vision for K-12 CS is to:

1. Introduce the fundamental concepts of CS to all students, beginning at the elementary school level;
2. Present CS at the secondary school level in a way that will be both accessible and worthy of a CS credit, or as a core graduation credit;
3. Offer additional secondary-level CS standards that will allow interested students to study facets of CS in depth and prepare them for entry into a career or college; and
4. Increase the knowledge of CS for all students, especially those from underrepresented groups in this field.

Wisconsin's Approach to Academic Standards for Computer Science

With the release of the *Wisconsin Standards for Computer Science* (CS), Wisconsin CS teachers have access to the foundational knowledge and skills needed to educate students for successful entry into hundreds of high-wage, high-demand occupations and careers. Vetted by business, industry and education professionals, these academic standards guide Wisconsin schools, teachers, and community partners toward development and continuous improvement of world-class CS courses.

The Computer Science Teachers Association (CSTA) is a professional organization that supports and promotes the teaching of CS. The *2011 CSTA K–12 CS Standards* represented the consensus view across the computing profession, educators, and academia. The next community revision of the CSTA standards are expected in mid-2017, but our work in Wisconsin is informed by an interim draft made available during 2016, as well as a separate but related *K-12 Computer Science Framework* under development with the involvement of many other states. The *Wisconsin Standards for Computer Science* share five overall conceptual strands with these previous standards documents. The learning priorities and performance indicators contained within each set of CS standards consists of knowledge and skills specific to each of the five strands:

- Algorithms and Programming,
- Computing Systems,
- Data and Analysis,
- Impacts of Computing, and
- Networks and the Internet.

These are, of course, critical as students develop an understanding of CS as a discipline as well as how these skills intersect with other content areas. In addition, there are many knowledge areas, skills, and dispositions delineated in these CS academic standards that are common to the pursuit of careers and postsecondary education in many fields.

Numerous existing sets of standards and standards-related documents have been used in developing the Wisconsin Standards for Computer Science. These include:

- The (Interim) CSTA K-12 Computer Science Standards, revised 2016 http://www.csteachers.org/?page=CSTA_Standards
- The K-12 Computer Science Framework https://k12cs.org/
- Approved or draft standards from the following states:
    - Arkansas: http://www.arkansased.gov/divisions/learning-services/curriculum-and-instruction/curriculum-framework-documents/computer-science
    - Florida: http://www.cpalms.org/Public/search/Standard (under Science K-2, 3-5, 6-8, 9-12)
    - Idaho (Draft): http://www.sde.idaho.gov/topics/content-standards/index.html
    - Indiana: http://www.doe.in.gov/standards/science-computer-science
    - Massachusetts: http://www.doe.mass.edu/stem/standards.html
    - New Jersey: http://www.state.nj.us/education/cccs/2014/tech/82.pdf
    - South Carolina (draft): http://ed.sc.gov/instruction/standards-learning/computer-science/
    - Texas: http://ritter.tea.state.tx.us/rules/tac/chapter126/index.html
    - Washington: http://www.k12.wa.us/ComputerScience/LearningStandards.aspx

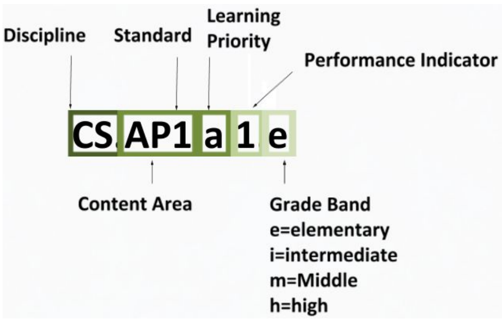
The *Wisconsin Standards for Computer Science* may be taught and integrated through a variety of classes and experiences. Each district, school, and program area should determine the means by which students meet these standards. Through the collaboration of multiple stakeholders, these foundational standards will set the stage for high-quality, successful, contemporary CS courses and programs throughout Wisconsin's PK-12 systems.

Standard Structure

The Wisconsin Standards for Computer Science follow a specific structure.

Discipline: Computer Science (CS)				
Content Area: Algorithms and Programming (AP)				
Standard: AP1: Students will recognize and define computational problems using algorithms and programming.				
	Performance Indicators (By Grade Band)			
Learning Priority	**K-2**	**3-5**	**6-8**	**9-12**
AP1.a: Develop algorithms.	AP1.a.1.e: Construct and execute algorithms (sets of step-by-step instructions), which include sequencing, and simple loops to accomplish a task, both independently and collaboratively, with or without a computing device.	AP1.a.4.i: Construct and execute algorithms (sets of step-by-step instructions), which include sequencing, loops, and conditionals to accomplish a task, both independently and collaboratively, with or without a computing device.	AP1.a.6.m: Decompose a computational problem into parts and create solutions for one or more parts.	AP1.a.8.h: Analyze a problem, and then design and implement an algorithmic solution using sequence, selection and iteration.
	AP1.a.2.e: Decompose (break down) a larger computational problem into smaller sub-problems independently or with teacher	AP1.a.5.i: Decompose (break down) a larger computational problem into smaller sub-problems independently or in a collaborative group.	AP1.a.7.m: Identify how subproblems could be recombined to create something new (e.g., break down the individual parts that would be needed to	AP1.a.9.h: Explain and demonstrate how modeling and simulation can be used to explore natural phenomena (e.g., flocking behaviors, queueing, life cycles).

Standard Coding



Standard Formatting

- **Standard**: Broad statement that tells what students are expected to know or be able to do.
- **Learning Priority**: Breaks down the broad statement into manageable learning pieces.
- **Performance Indicator by grade band**: Measurable degree to which a standard has been developed or met.

Grade Bands

Grade bands of K-2, 3-5, 6-8, and 9-12 align to typical elementary, middle, and high school levels.

- Grade band K-2 and 3-5 performance indicators represent knowledge and skills that should be integrated throughout the elementary curriculum.
- Computer Science education should be part of the core curriculum for all middle school students. Awareness, exploration, and building foundational skills should occur in middle school.
- Computer Science education at the high school level continues to develop student foundational understanding of CS in the world through in-depth CS learning, including awareness and exploration activities. Performance indicators marked with a (+) for grades 9-12 represent advanced CS learning expectations for students with aspirations toward careers and postsecondary studies in computing disciplines.

Content Area: Algorithms and Programming (AP)

Discipline: Computer Science (CS)

Content Area: Algorithms and Programming (AP)

Standard: AP1: Students will recognize and define computational problems using algorithms and programming.

	Performance Indicators (By Grade Band)			
Learning Priority	**K-2**	**3-5**	**6-8**	**9-12**
AP1.a: Develop algorithms.	AP1.a.1.e: Construct and execute algorithms (sets of step-by-step instructions), which include sequencing, and simple loops to accomplish a task, both independently and collaboratively, with or without a computing device.	AP1.a.4.i: Construct and execute algorithms (sets of step-by-step instructions), which include sequencing, loops, and conditionals to accomplish a task, both independently and collaboratively, with or without a computing device.	AP1.a.6.m: Decompose a computational problem into parts and create solutions for one or more parts.	AP1.a.8.h: Analyze a problem, and then design and implement an algorithmic solution using sequence, selection and iteration.
	AP1.a.2.e: Decompose (break down) a larger computational problem into smaller sub-problems independently or with teacher	AP1.a.5.i: Decompose (break down) a larger computational problem into smaller sub-problems independently or in a collaborative group.	AP1.a.7.m: Identify how subproblems could be recombined to create something new (e.g., break down the individual parts that would be needed to	AP1.a.9.h: Explain and demonstrate how modeling and simulation can be used to explore natural phenomena (e.g., flocking behaviors, queueing, life cycles).

	guidance. (e.g., to draw a snowman, we can draw several different, simpler shapes).		program a certain type of game and then show how the parts could be reused in other types of games).	
	AP1.a.3.e: Categorize a group of items based on the attributes or actions of each item, with or without a computing device.			AP1.a.10.h: (+) Provide examples of computationally solvable problems and difficult-to-solve problems.
				AP1.a.11.h: (+) Decompose a large-scale computational problem by identifying generalizable patterns and applying them in a solution.
				AP1.a.12.h: (+) Illustrate the flow of execution of a recursive algorithm.
				AP1.a.13.h: (+) Describe how parallel processing can be used to solve large computational problems (e.g., SETI at Home, FoldIt).

				AP1.a.14.h: (+) Develop and use a series of test cases to verify that a program performs according to its design specifications.
				AP1.a.15.h: (+) Explain the value of heuristic algorithms (discovery methods) to approximate solutions for difficult to solve computational problems.

Standard: AP2: Students will create computational artifacts using algorithms and programming.				
AP2.a Develop and implement an artifact.	AP2.a.1.e: Construct programs to accomplish a task or as a means of creative expression, which include sequencing, events and simple loops, using a block-based visual programming language, both independently and collaboratively (e.g., pair programming).	AP2.a.3.i: Construct programs in order to solve a problem or for creative expression, which include sequencing, events, loops, conditionals, parallelism and variables, using a block-based visual programming language or text based language, both independently and collaboratively (e.g., pair programming).	AP2.a.6.m: Develop programs, both independently and collaboratively, which include sequencing with nested loops and multiple branches [Clarification: At this level, students may use block-based and/or text-based languages].	AP2.a.10.h: Use user-centered research and design techniques (e.g., surveys, interviews) to create software solutions.
	AP2.a.2.e: Plan and create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, sequential graphic organizer).	AP2.a.4.i: Create a plan as part of the iterative design process, both independently and with diverse collaborative teams (e.g., storyboard, flowchart, pseudo-code, story map).	AP2.a.7.m: Produce computational artifacts with broad accessibility and usability through careful consideration of diverse needs and wants of the community.	AP2.a.11.h: Integrate grade-level appropriate mathematical techniques, concepts, and processes in the creation of computational artifacts.

		AP2.a.5.i: Use mathematical operations to change a value stored in a variable.	AP2.a.8.m: Use an iterative design process (e.g., define the problem, generate ideas, build, test, and improve solutions) to solve computational problems, both independently and collaboratively.	AP2.a.12.h: Design, develop, and implement a computing artifact that responds to an event (e.g., robot that responds to a sensor, mobile app that responds to a text message, sprite that responds to a broadcast).
			AP2.a.9.m: Create variables that represent different types of data and manipulate their values.	AP2.a.13.h: (+) Decompose a computational problem by creating new data types, functions or classes.
				AP2.a.14.h: (+) Develop programs for multiple computing platforms (e.g., computer desktop, web, mobile).
				AP2.a.15.h: (+) Implement an Artificial Intelligence (AI) algorithm to play a game against a human opponent or solve a problem.

				AP2.a.16.h: (+) Demonstrate code reuse by creating programming solutions using libraries and Application Program Interfaces (APIs). (e.g., graphics libraries, maps, API).

Standard: AP3: Students will communicate about computing ideas.				
AP3.a Recognize and cite sources.	AP3.a.1.e: Give credit to the source when using code, music, or pictures that were created by others.	AP3.a.2.i: Use proper citations and document when ideas are borrowed and changed for their own use (e.g., using pictures created by others, using music created by others, remixing programming projects).	AP3.a.3.m: Provide proper attribution when code is borrowed or built upon.	AP3.a.4.h: Compare and contrast various software licensing schemes (e.g., open source, freeware, commercial).
AP3.b Communicate about technical and social issues.	AP3.b.1.e: Follow simple instructions to complete a task, such as a simple visual tutorial.	AP3.b.2.i: Understand that algorithms have impacted society in both beneficial and harmful ways.	AP3.b.5.m: Discuss how algorithms have impacted society - both the beneficial and harmful effects.	AP3.b.8.h: Evaluate and analyze how algorithms have impacted our society and discuss the benefits and harmful impacts of a variety of technological innovations.
		AP3.b.3.i: Compare different problem solving techniques.	AP3.b.6.m: Compare different algorithms that may be used to solve the same problem in terms of their speed, clarity and size (e.g., different algorithms solve the same problem, but one might be faster than the	AP3.b.9.h: (+) Compare a variety of programming languages and identify features that make them useful for solving different types of problems and developing different kinds of systems (e.g., declarative, logic,

			other). [Clarification: Students are not expected to quantify these differences].	parallel, functional, compiled, interpreted, real-time).
		AP3.b.4.i: Modify a set of instructions (e.g., in dancing, cooking or other areas) and discuss how many paths can lead to the same result.	AP3.b.7.m: Modify existing code to change its functionality, and discuss the variety of ways in which to do this.	AP3.b.10.h: (+) Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality).
AP3.c Document code.			AP3.c.1.m: Interpret the flow of execution of algorithms and predict their outcomes. [Clarification: Algorithms can be expressed using natural language, flow and control diagrams, comments within code, and pseudocode].	AP3.c.3.h: (+) Describe how Artificial Intelligence (AI) drives many software and physical systems (e.g., autonomous robots, computer vision, pattern recognition, text analysis).
			AP3.c.2.m: Use documentation regarding code to modify programs.	AP3.c.4.h: Write appropriate documentation for programs.

				AP3.c.5.h: (+) Use application programming interface (APIs) documentation resources.
				AP3.c.6.h: Use online resources to answer technical questions.

Standard: AP4: Students will develop and use abstractions.				
AP4.a Create and use abstractions (representations) to solve complex computational problems.	AP4.a.1.e: Use numbers or other symbols to represent data (e.g., thumbs up/down for yes/no, color by number, arrows for direction, encoding/decoding a word using numbers or pictographs).	AP4.a.2.i: Use several existing functions/procedures to solve a problem (e.g., using several square, circle, and triangle drawing functions to create a larger picture).	AP4.a.3.m: Define and use functions/procedures that hide the complexity of a task and can be reused to solve similar tasks. [Clarification: Students use and modify, but do not necessarily create, functions/procedures with parameters].	AP4.a.4.h: Demonstrate the value of abstraction for managing problem complexity (e.g., using a list instead of discrete variables).
				AP4.a.5.h: Understand the notion of hierarchy and abstraction in high-level languages, translation, instruction sets, and logic circuits.
				AP4.a.6.h : Deconstruct a complex problem into simpler parts using predefined constructs (e.g., functions and parameters and/or classes).

				AP4.a.7.h: (+) Compare and contrast fundamental data structures and their uses (e.g., lists, maps, arrays, stacks, queues, trees, graphs).
				AP4.a.8.h: (+) Critically analyze and evaluate classic algorithms (e.g., sorting, searching) and use in different contexts, adapting as appropriate.
				AP4.a.9.h: (+) Discuss issues that arise when breaking large-scale problems down into parts that must be processed simultaneously on separate systems (e.g., cloud computing, parallelization, concurrency).
				AP4.a.10.h: (+) Define the functionality of an abstraction without implementing the abstraction.

				AP4.a.11.h: (+) Evaluate algorithms (e.g., sorting, searching) in terms of their efficiency, correctness and clarity.
				AP4.a.12.h: (+) Identify programming language features that can be used to define or specify an abstraction.
				AP4.a.13.h: (+) Identify abstractions used in a solution (program or software artifact) and reuse those abstractions to solve a different problem.

Standard: AP5: Students will collaborate with diverse teams.				
AP5.a Work together to solve computational problems using a variety of resources.	AP5.a.1.e: Work together with a team to create a solution to a computational problem.	AP5.a.3.i: Apply collaboration strategies to support problem solving within the design cycle of a program.	AP5.a.5.m: Solicit and integrate peer feedback as appropriate to develop or refine a program.	AP5.a.6.h: Design and develop a software artifact working in a team.
	AP5.a.2.e: Use teachers, parents, and other resources to solve a computational problem.	AP5.a.4.i: Understand there are many resources that can be used/tapped to solve a problem.		AP5.a.7.h: Demonstrate how diverse collaborating impacts the design and development of software products (e.g., discussing real-world examples of products which have been improved through having a diverse design team or reflecting on their own team's development experience).
				AP5.a.8.h: (+) Demonstrate software life cycle processes (e.g., spiral, waterfall) by participating on software project teams (e.g., community service project with real-world clients).

				AP5.a.9.h: (+) Use version control systems, Integrated Development Environments (IDEs), and collaboration tools and practices (code documentation) in a group software project.
AP5.b Foster an inclusive computing culture.	AP5.b.1.e: Understand the value for teams of including members with different perspectives, experiences, and backgrounds, including race, gender, ethnicity, language, disability, family background, and family income.		AP5.b.2.m: Analyze team members’ strengths and utilize them to foster an inclusive computing culture.	AP5.b.3.h: Create design teams taking into account the strengths and perspectives of potential team members.

Standard: AP6: Students will test and refine computational solutions.				
AP6.a Test and debug computational solutions.	AP6.a.1.e: Analyze and debug (fix) an algorithm, which includes sequencing and simple loops, with or without a computing device.	AP6.a.2.i: Analyze and debug (fix) an algorithm, which includes sequencing, events, loops, conditionals, parallelism, and variables.	AP6.a.3.m: Use testing and debugging methods to ensure program correctness and completeness.	AP6.a.4.h: Use a systematic approach and debugging tools to independently debug a program (e.g., setting breakpoints, inspecting variables with a debugger).
AP6.b Develop and apply success criteria.		AP6.b.1.i: Determine the correctness of a computational problem solution by listening to a classmate describe the solution.	AP6.b.2.m: Apply a rubric to determine if and how well a program meets objectives.	AP6.b.3.h: (+) Evaluate key qualities of a program (e.g., correctness, usability, readability, efficiency, portability, scalability) through a process such as a code review.

Content Area: Computing Systems (CS)

Discipline: Computer Science (CS)

Content Area: Computing Systems (CS)

Standard: CS1: Students will communicate about computing systems.

	Performance Indicators (By Grade Band)			
Learning Priority	**K-2** (e)	**3-5 (i)**	**6-8** (m)	**9-12** (h)
CS1.a Identify hardware and software components.	CS1.a.1.e: Identify and use software that controls computational devices to accomplish a task (e.g., use an app to draw on the screen, use software to write a story or control robots).	CS1.a.3.i: Select and operate appropriate software to perform a variety of tasks and recognize that users have different needs and preferences for the technology they use.	CS1.a.5.m: Justify the suitability of hardware and software chosen to accomplish a task (e.g., comparison of the features of a tablet vs. desktop, selecting which sensors and platform to use in building a robot or developing a mobile app).	CS1.a.6.h: Develop and apply criteria (e.g., power consumption, processing speed, storage space, battery life, cost, operating system) for evaluating a computer system for a given purpose (e.g., system specification needed to run a game, web browsing, graphic design or video editing).
	CS1.a.2.e: Use appropriate terminology in naming and describing the function of common computing devices and	CS1.a.4.i: Use appropriate terminology in naming internal and external components of computing devices and		CS1.a.7.h: (+) Identify the functionality of various categories of hardware components and communication

	components (e.g., desktop computer, laptop computer, tablet device, monitor, keyboard, mouse, printer).	describing their relationships, capabilities, and limitations.		between them (e.g., physical layers, logic gates, chips, input and output devices).
CS1.b Understand how the components of a computer system work together.	CS1.b.1.e: Identify the components of a computer system and what the basic functions are (e.g., hard drive, and memory) as well as external features and their uses (e.g., printers, scanners, external hard drives, and cloud storage).	CS1.b.2.i: Model how a computer system works. [Clarification: Only includes basic elements of a computer system, such as input, output, processor, sensors, and storage].		CS1.b.3.h: (+) Explain the role of operating systems (e.g., how programs are stored in memory, how data is organized/retrieved, how processes are managed and multi-tasked).

Standard: CS2: Students will test and refine computing systems.				
CS2.a Problem solve and debug.	CS2.a.1.e: Identify, using accurate terminology, simple hardware and software problems that may occur during use (e.g., app or program not working as expected, no sound, device won't turn on).	CS2.a.2.i: Identify, using accurate terminology, simple hardware and software problems that may occur during use, and apply strategies for solving problems (e.g., reboot device, check for power, check network availability, close and reopen app).	CS2.a.3.m: Use a systematic process to identify the source of a problem within individual and connected devices (e.g., follow a troubleshooting flow diagram, make changes to software to see if hardware will work, restart device, check connections, swap in working components).	CS2.a.4.h: Devise a systematic process to identify the source of a problem within individual and connected devices (e.g., research, investigate, problem solve).

Standard: CS3: Students will develop and use abstractions in computing systems.				
CS3.a Generalize in computer systems.			CS3.a.1.m: Analyze the relationship between a device's computational components and its capabilities. (e.g., computing systems include not only computers, but also cars, microwaves, smartphones, traffic lights, and flash drives).	CS3.a.2.h: Demonstrate the role and interaction of a computer embedded within a physical system, such as a consumer electronic, biological system, or vehicle, by creating a diagram, model, simulation, or prototype.
				CS3.a.3.h: (+) Describe the steps necessary for a computer to execute high-level source code (e.g., compilation to machine language, interpretation, fetch-decode-execute cycle).

Standard: CS4: Students will create and modify computing systems.				
CS4.a Modify and create computational artifacts.			CS4.a.1.m: Extend or modify existing programs to add simple features and behaviors using different forms of inputs and outputs (e.g., inputs such as sensors, mouse clicks, data sets; outputs such as text, graphics, sounds).	CS4.a.2.h: Create, extend, or modify existing programs to add new features and behaviors using different forms of inputs and outputs (e.g., inputs such as sensors, mouse clicks, data sets; outputs such as text, graphics, sounds).
				CS4.a.3.h: (+) Create a new artifact that uses a variety of forms of inputs and outputs (e.g., inputs such as sensors, mouse clicks, data sets; outputs such as text, graphics, sounds).

Content Area: Data and Analysis (DA)

Discipline: Computer Science (CS)				
Content Area: Data and Analysis (DA)				
Standard: DA1: Students will create computational artifacts using data and analysis.				
	Performance Indicators (By Grade Band)			
Learning Priority	**K-2**	**3-5**	**6-8**	**9-12**
DA1.a Represent and manipulate data.		DA1.a.1.i: Use numeric values to represent non-numeric ideas in the computer (e.g., binary, American Standard Code for Information Interchange (ASCII), pixel attributes such as Red Green Blue (RGB)).	DA1.a.3.m: Represent data using different encoding schemes (e.g., binary, Unicode, Morse code, shorthand, student-created codes).	DA1.a.4.h: Convert between binary, decimal, and hexadecimal representations of data (e.g., convert hexadecimal color codes to decimal percentages, ASCII/Unicode representation).
		DA1.a.2.i: Answer a question by using a computer to manipulate (e.g., sort, total and/or average, chart, graph) and analyze data that has been collected by the class or student.		DA1.a.5.h: Analyze the representation tradeoffs among various forms of digital information (e.g., lossy vs. lossless compression, encrypted vs. unencrypted, various image representations).

				DA1.a.6.h: (+) Discuss how data sequences (e.g., binary, hexadecimal, octal) can be interpreted in a variety of forms (e.g., instructions, numbers, text, sound, image).

Standard: DA2: Students will recognize and define data in computational problems.

DA2.a Gather data to support computational problem solving.	DA2.a.1.e: Collect simple quantitative data over time (e.g., daily temperatures or sunrise time).	DA2.a.2.i: Collect quantitative data over time from multiple sources (e.g., class or group pools individual observations of street traffic).	DA2.a.3.m: Gather and organize multiple quantitative data elements using a computational tool (e.g., spreadsheet software).	DA2.a.4.h: Discuss techniques used to store, process, and retrieve different amounts of information (e.g., files, databases, data warehouses).
				DA2.a.5.h: (+) Use various data collection techniques for different types of computational problems (e.g., mobile device Global Positioning System (GPS), user surveys, embedded system sensors, open data sets, social media data sets).
DA2.b Categorize and analyze data.	DA2.b.1.e: Sort objects into buckets, recognizing relevant and/or irrelevant data (e.g., one of these things is not like the other).	DA2.b.2.i: Choose appropriate classifications or grouping for data by shape, color, size, or other attributes.	DA2.b.3.m: Develop a strategy to answer a question by using a computer to manipulate (e.g., sort, total and/or average, chart, graph) and analyze data that has been collected by the class or student.	DA2.b.4.h: Apply basic techniques for locating and collecting small- and large-scale data sets (e.g., creating and distributing user surveys, accessing real-world data sets).

Standard: DA3: Students will communicate about data in computing.				
DA3.a Communicate about data.	DA3.a.1.e: Collect data over time and organize it in a chart or graph in order to make and communicate a prediction.	DA3.a.2.i: Organize data into new subsets to provide different views or commonalities and present insights gained using visual representations.	DA3.a.4.m: Describe how different formats of stored data represent tradeoffs between quality and size. [Clarification: compare examples of music, text and/or image formats].	DA3.a.6.h: Use computational tools to collect, transform, and organize data about a problem to explain to others.
		DA3.a.3.i: Organize and evaluate data for its sufficiency and relevance to making accurate inferences or predictions.	DA3.a.5.m: Explain the processes used to collect, transform, and analyze data to solve a problem using computational tools (e.g., use an app or spreadsheet form to collect data, decide which data to use or ignore, and choose a visualization method).	

Standard: DA4: Students will develop and use data abstractions.				
DA4.a Model with data.	DA4.a.1.e: Use a computing device to store, search, retrieve, modify, and delete information and define the information stored as data.	DA4.a.3.i: Create a computational artifact to model the attributes and behaviors associated with a concept (e.g., solar system, life cycle of a plant).	DA4.a.4.m: Revise computational models to more accurately reflect real-world systems (e.g., ecosystems, epidemics, spread of ideas).	DA4.a.6.h: Create computational models that simulate real-world systems (e.g., ecosystems, epidemics, spread of ideas).
	DA4.a.2.e: Create a model of an object or process in order to identify patterns and essential elements (e.g., water cycle, butterfly life cycle, seasonal weather patterns).		DA4.a.5.m: Modify an existing computational model to emphasize key features and relationships within a system. (A model can be used to simulate events, examine theories and inferences, or make predictions).	DA4.a.7.h: (+) Evaluate the ability of models and simulations to formulate, refine, and test hypotheses.
DA4.b Identify patterns.				DA4.b.1.h:(+) Use data analysis to identify significant patterns in complex systems (e.g., take existing data sets and make sense of them).
				DA4.b.2.h: (+) Identify mathematical and computational patterns

				through modeling and simulation (e.g., regression, queueing theory, discrete event simulation).

Content Area: Impacts of Computing (IC)

Discipline: Computer Science (CS)				
Content Area: Impacts of Computing (IC)				
Standard: IC1: Students will understand the impact and effect computing technology has on our everyday lives.				
	Performance Indicators (By Grade Band)			
Learning Priority	**K-2**	**3-5**	**6-8**	**9-12**
IC1.a Understand the impact technology has on our everyday lives, and the effects of computing on the economy and culture.	IC1.a.1.e: Compare and contrast examples of how computing technology has changed the way people live, work, and interact.	IC1.a.2.i: Discuss computing technologies that have changed the world and express how those technologies influence, and are influenced by, cultural practices.	IC1.a.4.m: Provide examples of how computational artifacts and devices impact health and wellbeing, both positively and negatively, locally and globally (e.g., effects of globalization, and automation).	IC1.a.6.h: Debate the social and economic implications associated with ethical and unethical computing practices (e.g., intellectual property rights, hacktivism, software piracy, new computers shipped with malware).
		IC1.a.3.i: Generate examples of how computing can affect society, and also how societal values can shape computing choices.	IC1.a.5.m: Explain how computer science fosters innovation and can enhance careers and disciplines.	IC1.a.7.h: Discuss implications of the collection and large-scale analysis of information about individuals (e.g., how businesses, social media, and government collect and use personal data).

				IC1.a.8.h: Compare and debate the positive and negative impacts of computing on behavior and culture (e.g., evolution from hitchhiking to ride-sharing apps, online accommodation rental services).
				IC1.a.9.h: Describe how computation shares features with art and music by translating human intention into an artifact.
				IC1.a.10.h: (+) Develop criteria to evaluate the beneficial and harmful effects of computing innovations on people and society.
IC1.b Understand the effects of computing on communication and relationships.	IC1.b.1.e: Explain the differences between communicating electronically and communicating in person.	IC1.b.2.i: Compare and contrast the effects of communicating electronically to communicating in person.	IC1.b.3.m: Analyze and present beneficial and harmful effects of personal electronic communication and social electronic communication.	IC1.b.5.h: Evaluate the negative impacts of electronic communication on personal relationships and evaluate differences between face-to-face and electronic communication.

			IC1.b.4.m: Describe ways in which the Internet impacts global communication and collaborating.	IC1.b.6.h: (+) Create a list of practices that individuals and organizations can use to encourage proper use of both electronic and face-to-face communication.
				IC1.b.7.h: (+) Evaluate the negative impacts on societal discourse caused by social media and electronic communities.

Standard: IC2: Students will experience learning within a collaborative, inclusive computing culture and explain the steps needed to ensure that all people have access to computing.				
IC2.a Understand the effects of the digital divide.		IC2.a.1.i: Brainstorm and advocate for ways in which computing devices and the Internet could be made more available to all people.	IC2.a.2.m: Explain the impact of the digital divide (i.e., uneven access to computing, computing education, and interfaces) on access to critical information.	IC2.a.3.h: (+) Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society.
IC2.b Test and refine digital artifacts for accessibility		IC2.b.1.i: Brainstorm ways in which computing devices could be made more accessible to all users.	IC2.b.2.m: Critically evaluate and redesign a computational artifact to remove barriers to universal access (e.g., using captions on images, high contrast colors, and/or larger font sizes).	IC2.b.3.h: Design a user interface (e.g., web pages, mobile applications, animations) to be more inclusive, accessible, minimizing the impact of the designer's inherent bias.
IC2.c Collaborate ethically in the creation of digital artifacts.	IC2.c.1.e: Work with others as co-learners to solve a problem, or reach a goal.	IC2.c.2.i: Use online collaborative spaces ethically and safely to work with another student to solve a problem or reach a goal.	IC2.c.4.m: Use the Internet ethically and safely to work with a group of people who are not physically near to solve a problem or reach a goal.	IC2.c.5.h: Ethically and safely select, observe, and contribute to global collaboration in the development of a computational artifact (e.g., contribute the resolution of a bug in an open-source project platform, or contribute an online article).

		IC2.c.3.i: Seek out and compare diverse perspectives, synchronously or asynchronously, to improve a project.		IC2.c.6.h: Demonstrate how computing enables new forms of experience, expression, communication, and collaboration.

Standard: IC3: Students will understand the importance of proper use of data and information in a computing society.				
IC3.a Understand intellectual property and fair use.		IC3.a.1.i: Use resources from the World Wide Web in making artifacts and recognize that the work came from others.	IC3.a.2.m: Understand laws associated with digital information (e.g., intellectual property, fair use, and Creative Commons).	IC3.a.4.h: Compare and contrast information access and distribution rights.
			IC3.a.3.m: Describe ethical issues that relate to computing devices and networks (e.g., equity of access, security, hacking, intellectual property, copyright, Creative Commons licensing, and plagiarism).	
IC3.b Assess the practice of digital privacy.	IC3.b.1.e: Respect other students' information and refrain from accessing others' devices or accounts without permission.	IC3.b.3.i: Explain problems that relate to using computing devices and networks (e.g., logging out to deter others from using your account, cyberbullying, privacy of personal information, and ownership).	IC3.b.4.m: Analyze and summarize negative and positive impacts of using data and information to categorize people, predict behavior, and make recommendations based on those predictions (e.g.,	IC3.b.5.h: Research and understand misuses of private digital information in our society.

			customizing search results or targeted advertising, based on previous browsing history, can save search time and limit options at the same time).	
	IC3.b.2.e: Understand what kinds of digital information is considered private, take steps to keep their information private, and respect the privacy of other students' information.			IC3.b.6.h: Debate laws regarding an individual's digital privacy, and be able to explain the main arguments from multiple perspectives.
IC3.c Assess interrelationship between computing and society.				IC3.c.1.h: (+) Design and implement a study that evaluates how computation has revolutionized an aspect of our culture, or predicts how an aspect might evolve (e.g., education, healthcare, art/entertainment, energy).

				IC3.c.2.h: (+) Debate laws and regulations that impact the development and use of software, and be able to explain the main arguments from multiple perspectives.

Content Area: Networking and the Internet (NI)

Discipline: Computer Science (CS)				
Content Area: Networking and the Internet (NI)				
Standard: NI1: Students will understand the importance of security when using technology.				
	Performance Indicators (By Grade Band)			
Learning Priority	**K-2**	**3-5**	**6-8**	**9-12**
NI1.a Use secure practices for personal computing.	NI1.a.1.e: Use secure practices (such as passwords) to protect private information and discuss the effects of misuse.	NI1.a.2.i: Create examples of strong passwords, explain why strong passwords should be used, and demonstrate proper use and protection of personal passwords.	NI1.a.4.m: Analyze and summarize security risks associated with weak passwords, lack of encryption, insecure transactions, and persistence of data.	NI1.a.6.h: Provide examples of personal data that should be kept secure and the methods by which individuals keep their private data secure.
		NI1.a.3.i: Remember basic concepts/facts regarding security issues with general computer use.	NI1.a.5.m: Understand security issues with general computer use.	NI1.a.7.h: (+) Explain security issues that might lead to compromised computer programs (e.g., circular references, ambiguous program calls, lack of error checking, and field size checking).

NI1.b Understand the importance of institutional security.		NI1.b.1.i: Give examples of information that organizations keep private as opposed to information that they make public.	NI1.b.2.m: Explain the principles of information security (confidentiality, integrity, availability) and authentication techniques.	NI1.b.3.h: Compare and contrast multiple viewpoints on cybersecurity (e.g., from the perspective of security experts, privacy advocates, national security).
				NI1.b.4.h: Identify digital and physical strategies to secure networks and discuss the tradeoffs between ease of access and need for security.

Standard: NI2: Students will understand how information is sent by the Internet.				
NI2.a Demonstrate how the Internet works at the physical layer.	NI2.a.1.e: Use a physical tool (e.g. flashlight, string) to communicate with another student.	NI2.a.3.i: Model how a device on a network sends a message from one device (sender) to another (receiver) while following specific rules.	NI2.a.6.m: Simulate how information is transmitted as packets through multiple devices over the Internet and networks.	NI2.a.8.h: Illustrate the basic components of computer networks (e.g., draw logical and topological diagrams of networks including routers, switches, servers, and end user devices; create model with string and paper).
	NI2.a.2.e: Provide examples of computer use that involve the Internet.	NI2.a.4.i: Differentiate between using the Internet and not using the Internet (e.g. identify difference between local and remote computation, such as collaborating on a Google Doc in "the cloud" versus editing a local document).	NI2.a.7.m: Explain, using basic terms, how a wireless or cellular network allows Internet information to be transmitted from a server to a user device.	NI2.a.9.h: (+) Explain ways in which the Internet is decentralized and fault-tolerant.
		NI2.a.5.i: Illustrate how information travels on the Internet.		NI2.a.10.h: (+) Simulate and discuss the issues (e.g., bandwidth, load, delay, topology) that impact network

				functionality (e.g., use free network simulators).
NI2.b Demonstrate how the Internet works at the protocol layer.		NI2.b.1.i: Act out a protocol that people use in common everyday communications (e.g., checking out a book from the library, meeting a new person, making an appointment, playing a class game, or calling a friend on the phone to invite them over).	NI2.b.2.m: Define the term protocol, provide an example of protocols in daily life, and explain their use on the Internet.	NI2.b.3.h: Describe key protocols and underlying processes of Internet-based services (e.g., http/https and Simple Mail Transfer Protocol (SMTP)/Internet Message Access Protocol (IMAP), routing protocols).
NI2.c Demonstrate how the Internet works at the addressing layer.	NI2.c.1.e: Devise a system for sending a physical message to anyone in their school by using addressing techniques (e.g., address envelopes by student first name, and teacher, grade, or room).	NI2.c.2.i: Devise a system for sending a physical message to anyone in their school by using addressing techniques, and then draw a tree or visual representation of their addressing system, and finally act out their addressing system by sending messages.	NI2.c.3.m: Explain the hierarchical structure of the Internet Domain Name System.	NI2.c.4.h: (+) Evaluate how the hierarchical nature of the Domain Name System helps the Internet work efficiently.

NI2.d Demonstrate and explain encryption methods.		NI2.d.1.i: Communicate across a classroom using a secure method of their own design (e.g., pictures, physical movement, text).	NI2.d.2.m: Encode and decode text-based messages using basic algorithms (e.g., shift cipher, substitution cipher).	NI2.d.3.h: Write a program that performs basic encryption (e.g., shift cipher, substitution cipher).
				NI2.d.4.h: (+) Explain the features of public key cryptography.
				NI2.d.5.h: (+) Explore security policies by implementing and comparing encryption and authentication strategies (e.g., secure coding, safeguarding keys).